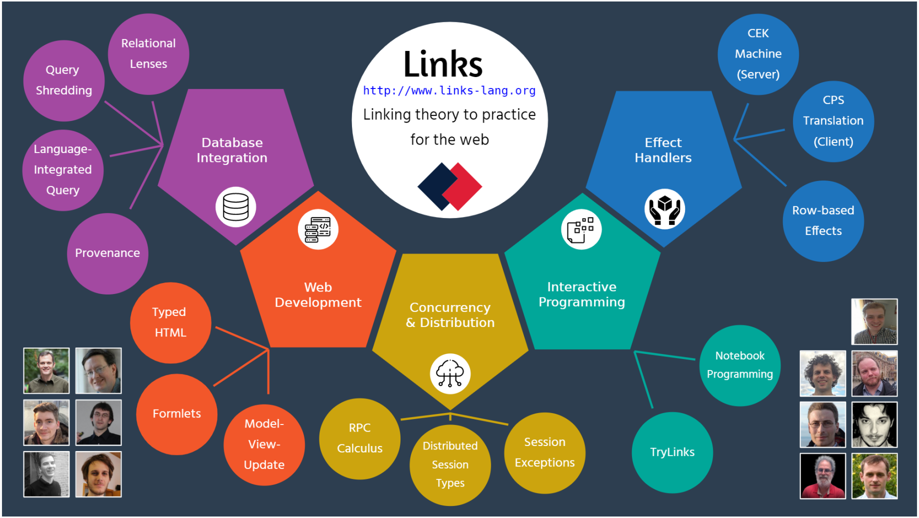
Tracking Linear Continuations for Effect Handlers

Wenhao Tang
The University of Edinburgh

SPLS, March 8th 2023

(Joint work with Sam Lindley, J. Garrett Morris, and Daniel Hillerström)

Links



Picture by Simon Fowler

Session Types in Links

Links session types:

- `!A.S` : send a value of type `A`, then continue as `S`
- `?A.S` : receive a value of type `A`, then continue as `S`
- `End` : no communication

Session Types in Links

Links session types:

- `!A.S` : send a value of type `A`, then continue as `S`
- `?A.S` : receive a value of type `A`, then continue as `S`
- `End` : no communication

Primitive operations on session-typed channels:

```
send    : ∀ a (b::Session) . (a, !a.b) → b
receive : ∀ a (b::Session) . (?a.b) → (a, b)
fork    : ∀   (a::Session) . (a → ()) → ~a
close   : End → ()
```

Session Types in Links are Sound

```
sig sender       : (!Int.End) → ()
fun sender(ch)   { var ch = send(42, ch); close(ch) }
```

Session Types in Links are Sound

```
sig sender        : (!Int.End) ⟶ ()
fun sender(ch)    { var ch = send(42, ch); close(ch) }
sig receiver      : (?Int.End) ⟶ ()
fun receiver(ch) { var (i, ch) = receive(ch); close(ch); printInt(i) }
```

Session Types in Links are Sound

```
sig sender        : (!Int.End) ⟶ ()
fun sender(ch)    { var ch = send(42, ch); close(ch) }
sig receiver      : (?Int.End) ⟶ ()
fun receiver(ch) { var (i, ch) = receive(ch); close(ch); printInt(i) }
```

```
links> { var ch = fork(receiver); sender(ch) };
42
```

Session Types in Links are Sound

```
sig sender        : (!Int.End) → ()
fun sender(ch)    { var ch = send(42, ch); close(ch) }
sig receiver      : (?Int.End) → ()
fun receiver(ch) { var (i, ch) = receive(ch); close(ch); printInt(i) }
```

```
links> { var ch = fork(receiver); sender(ch) };
42
```

```
links> { var ch = fork(receiver); var ch = send(42, ch); close(ch);
                                                          close(ch) };
<stdin>:1: Type error: Variable ch has linear type
    `End'
but is used 2 times.
In expression: var ch = send(42, ch);.
```

Session Types in Links are Sound

```
sig sender        : (!Int.End) ⟶ ()
fun sender(ch)    { var ch = send(42, ch); close(ch) }
sig receiver      : (?Int.End) ⟶ ()
fun receiver(ch) { var (i, ch) = receive(ch); close(ch); printInt(i) }
```

```
links> { var ch = fork(receiver); sender(ch) };
42
```

```
links> { var ch = fork(receiver); var ch = send(42, ch); close(ch);
                                  var ch = send(42, ch); close(ch) };
<stdin>:1: Type error: The function
    `send'
has type
    `(Int, !(Int).a::Session) ~b⟶ a::Session'
while the arguments passed to it have types
    `Int' and `End'
In expression: send(42, ch).
```

Session Types in Links are Sound

```
sig sender        : (!Int.End) → ()
fun sender(ch)    { var ch = send(42, ch); close(ch) }
sig receiver      : (?Int.End) → ()
fun receiver(ch) { var (i, ch) = receive(ch); close(ch); printInt(i) }
```

```
links> { var ch = fork(receiver); sender(ch) };
42
```

```
links> { var ch = fork(receiver);
         var f = fun(){ var ch = send(42, ch); close(ch) }; f(); f() };
<stdin>:1: Type error: Variable ch of linear type ~?(Int).End is used in a
    non-linear function literal.
In expression: fun(){var ch = send(42, ch); close(ch)}.
```

Session Types in Links are Sound

```
sig sender       : (!Int.End) → ()
fun sender(ch)   { var ch = send(42, ch); close(ch) }
sig receiver     : (?Int.End) → ()
fun receiver(ch) { var (i, ch) = receive(ch); close(ch); printInt(i) }
```

```
links> { var ch = fork(receiver); sender(ch) };
42
```

```
links> { var ch = fork(receiver);
         var f = linfun(){ var ch = send(42, ch); close(ch) }; f(); f() };
<stdin>:1: Type error: Variable f has linear type
    `() ~a~@ ()'
but is used 2 times.
In expression: var f = linfun(){var ch = send(42, ch); close(ch)};.
```

Effect Handlers in Links

```
sig choose : () { Choose: () ↠ Bool }⟶ ()
fun choose() { var i = if (do Choose) 42 else 1; printInt(i) }
```

Effect Handlers in Links

```
sig choose : () { Choose: () ↠ Bool }⟶ ()
fun choose() { var i = if (do Choose) 42 else 1; printInt(i) }
```

```
links> handle (choose())
         { case <Choose ⇒ r> ⟶ r(true) }
42
```

Effect Handlers in Links

```
sig choose : () { Choose: () ↠ Bool }⟶ ()
fun choose() { var i = if (do Choose) 42 else 1; printInt(i) }
```

```
links> handle (choose())
        { case <Choose ⇒ r> ⟶ r(true) }
42
```

```
links> handle (choose())
        { case <Choose ⇒ r> ⟶ r(true); r(false) }
42 1
```

Well Typed Programs Can Go Wrong in Links

```
sig sender2     : (!Int.End) { Choose: () ↠ Bool }⟶ ()
fun sender2(ch) { var i = if (do Choose) 42 else 1;
                  var ch = send(i, ch); close(ch) }
```

Well Typed Programs Can Go Wrong in Links

```
sig sender2     : (!Int.End) { Choose: () ↠ Bool }⟶ ()
fun sender2(ch) { var i = if (do Choose) 42 else 1;
                  var ch = send(i, ch); close(ch) }


links> handle ({ var ch = fork(receiver); sender2(ch) })
        { case <Choose ⇒ r> ⟶ r(true) }
42
```

Well Typed Programs Can Go Wrong in Links [1,2]

```
sig sender2     : (!Int.End) { Choose: () ↠ Bool }→ ()
fun sender2(ch) { var i = if (do Choose) 42 else 1;
                  var ch = send(i, ch); close(ch) }
```

```
links> handle ({ var ch = fork(receiver); sender2(ch) })
        { case <Choose ⇒ r> → r(true) }
42
```

```
links> handle ({ var ch = fork(receiver); sender2(ch) })
        { case <Choose ⇒ r> → r(true); r(false) }
***: Internal Error in evalir.ml (Please report as a bug): NotFound
 chan_7 (in Hashtbl.find) while interpreting.
```

[1] https://github.com/links-lang/links/issues/544
[2] Emrich and Hillerström, "Broken Links (Presentation)", 2020.

Our Main Contributions

- F°_{eff}: an extension of system F with correct interaction between linear types and effect handlers.
- Prove the safety of F°_{eff}.
- Q°_{eff}: a ML-variant of F°_{eff} with full type inference based on qualified types.

Our Main Contributions

- F°_{eff}: an extension of system F with correct interaction between linear types and effect handlers.
- Prove the safety of F°_{eff}.
- Q°_{eff}: a ML-variant of F°_{eff} with full type inference based on qualified types.

F° : System F with Linear Types[3]

Value types	$A, B ::= \alpha \mid A \to^{Y} B \mid \forall^{Y} \alpha^{K}.A$	Kinds $K ::=$
Linearity	$Y ::= \bullet \mid \circ$	Type^{Y}
Type contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$	
Kind contexts	$\Delta ::= \cdot \mid \Delta, \alpha : K$	
Terms	$M, N ::= x \mid \lambda^{Y} x^{A}.M \mid \Lambda^{Y} \alpha^{K}.M \mid M\, N \mid M\, A$	

[3]Mazurak, Zhao, and Zdancewic, “Lightweight Linear Types in System F^{o}”, 2010.

F° Examples

$$id = \Lambda^\bullet \alpha^{\mathsf{Type}^\circ}.\lambda^\bullet x^\alpha.x : \forall^\bullet \alpha^{\mathsf{Type}^\circ}.\alpha \to^\bullet \alpha$$

F° Examples

$$id = \Lambda^\bullet \alpha^{\mathsf{Type}^\circ}.\lambda^\bullet x^\alpha.x : \forall^\bullet \alpha^{\mathsf{Type}^\circ}.\alpha \to^\bullet \alpha$$

F° has subkinding $\mathsf{Type}^\bullet \le \mathsf{Type}^\circ$:

$$id\ Int\ 42 : Int$$

F° Examples

$$id = \Lambda^{\bullet}\alpha^{\mathsf{Type}^{\circ}}.\lambda^{\bullet}x^{\alpha}.x : \forall^{\bullet}\alpha^{\mathsf{Type}^{\circ}}.\alpha \to^{\bullet} \alpha$$

F° has subkinding $\mathsf{Type}^{\bullet} \leq \mathsf{Type}^{\circ}$:

$$id\ Int\ 42 : Int$$

Suppose we still have built-in session types, and omit the linearity annotations on terms and types when it is $\bullet$.

$$sendAndClose = \lambda f^{!Int.End}.\lambda^{\circ}x^{Int}.close\,(send\,(x, f)) : (!Int.End) \to Int \to^{\circ} ()$$

F_{eff} : System F with Effect Handlers[4]

		Kinds $K ::=$
Value types	$A, B ::= \alpha \mid A \to C \mid \forall \alpha^K.C$	Type
Computation types	$C, D ::= A \,!\, E$	Comp
Effect types	$E ::= \{R\}$	Effect
Row types	$R ::= \ell : P; R \mid \mu \mid \cdot$	$\mathsf{Row}_{\mathcal{L}}$
Presence types	$P ::= \mathsf{Abs} \mid A \twoheadrightarrow B \mid \theta$	Presence
Handler types	$F ::= C \rightrightarrows D$	Handler
Types	$T ::= A \mid R \mid P$	
Type contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$	
Kind contexts	$\Delta ::= \cdot \mid \Delta, \alpha : K$	
Values	$V, W ::= x \mid \lambda x^A.M \mid \Lambda \alpha^K.M$	
Computations	$M, N ::= V\,W \mid V\,T \mid (\mathbf{return}\ V)^E \mid (\mathbf{do}\ \ell\ V)^E$	
	$\mid\ \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \mid \mathbf{handle}\ M\ \mathbf{with}\ H$	
Handlers	$H ::= \{\mathbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H$	

[4] Hillerström, Lindley, and Atkey, "Effect handlers via generalised continuations", 2020.

$\mathrm{F_{eff}} + \mathrm{F}^{\circ}$ is BROKEN

Define $M; N \equiv \mathbf{let}\ \_ \leftarrow M\ \mathbf{in}\ N$.

Assuming a global channel $f : \mathit{End}$, we have:

$$\mathbf{handle}\ \overbrace{(\mathbf{do}\ \mathit{Choose}\ (); \mathit{close}\ f)}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow \mathit{Bool}\}}\ \mathbf{with}\ \overbrace{\{\mathit{Choose}\ \_\ \underset{\mathit{Bool}\to()\,!\,\{\}}{r} \mapsto r\ \mathit{true}; r\ \mathit{false}\}}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow \mathit{Bool}\}\rightrightarrows()\,!\,\{\}}$$

F_{eff} + F° is BROKEN

Define $M; N \equiv \mathbf{let}\ \_ \leftarrow M\ \mathbf{in}\ N$.

Assuming a global channel f : *End*, we have:

$$\mathbf{handle}\ \overbrace{(\mathbf{do}\ \mathit{Choose}\ (); \mathit{close}\ f)}^{()\,!\,\{\mathit{Choose}:() \twoheadrightarrow \mathit{Bool}\}}\ \mathbf{with}\ \overbrace{\{\mathit{Choose}\ \_\ \underset{\mathit{Bool} \to ()\,!\,\{\}}{r} \mapsto r\ \mathit{true}; r\ \mathit{false}\}}^{()\,!\,\{\mathit{Choose}:() \twoheadrightarrow \mathit{Bool}\} \rightrightarrows ()\,!\,\{\}}$$

$$\leadsto \quad (r\ \mathit{true}; r\ \mathit{false})[(\lambda\_.\mathit{close}\ f)/r]$$

$\mathrm{F}_{\mathrm{eff}} + \mathrm{F}^{\circ}$ is BROKEN

Define $M; N \equiv \mathbf{let}\ \_ \leftarrow M\ \mathbf{in}\ N$.

Assuming a global channel $f : \mathit{End}$, we have:

$$
\begin{aligned}
& \mathbf{handle}\ \overbrace{(\mathbf{do}\ \mathit{Choose}\ (); \mathit{close}\ f)}^{()\,!\,\{\mathit{Choose}:() \twoheadrightarrow \mathit{Bool}\}}\ \mathbf{with}\ \overbrace{\{\mathit{Choose}\ \_\ \underset{\mathit{Bool} \to ()\,!\,\{\}}{r} \mapsto r\ \mathit{true}; r\ \mathit{false}\}}^{()\,!\,\{\mathit{Choose}:() \twoheadrightarrow \mathit{Bool}\} \rightrightarrows ()\,!\,\{\}} \\
\rightsquigarrow\quad & (r\ \mathit{true}; r\ \mathit{false})[(\lambda\_.\mathit{close}\ f)/r] \\
=\quad & \mathit{close}\ f; \mathit{close}\ f
\end{aligned}
$$

f is closed twice!

$\mathrm{F}_{\mathrm{eff}} + \mathrm{F}^{\circ} = \mathrm{F}^{\circ}_{\mathrm{eff}}$

Value types	$A, B ::= \alpha \mid A \to^{Y} C \mid \forall^{Y} \alpha^{K}.C$
Computation types	$C, D ::= A \,!\, E$
Effect types	$E ::= \{R\}$
Row types	$R ::= \ell : P; R \mid \mu \mid \cdot$
Presence types	$P ::= \mathsf{Abs} \mid A \twoheadrightarrow^{Y} B \mid \theta$
Handler types	$F ::= C \rightrightarrows D$
Types	$T ::= A \mid R \mid P$
Label sets	$\mathcal{L} ::= \emptyset \mid \{\ell\} \uplus \mathcal{L}$
Linearity	$Y ::= \bullet \mid \circ$
Type contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$
Kind contexts	$\Delta ::= \cdot \mid \Delta, \alpha : K$
Values	$V, W ::= x \mid \lambda^{Y} x^{A}.M \mid \Lambda^{Y} \alpha^{K}.M$
Computations	$M, N ::= V\,W \mid V\,T \mid (\mathbf{return}\ V)^{E} \mid (\mathbf{do}\ \ell\ V)^{E}$
	$\mid\ \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \mid \mathbf{handle}\ M\ \mathbf{with}\ H$
Handlers	$H ::= \{\mathbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H$

Kinds $K ::=$

Type^{Y}

Comp

Effect

$\mathsf{Row}^{Y}_{\mathcal{L}}$

$\mathsf{Presence}^{Y}$

$\mathsf{Handler}$

Fixing $\mathrm{F_{eff}} + \mathrm{F}^\circ$

It would be great to know that r should be a linear function:

$$\textbf{handle}\ \overbrace{(\textbf{do}\ \mathit{Choose}\ (); \mathit{close}\ f)}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow\mathit{Bool}\}}\ \textbf{with}\ \overbrace{\{\mathit{Choose}\ \_\ \underset{\mathit{Bool}\to^\circ()\,!\,\{\}}{r} \mapsto r\ \mathit{true}; r\ \mathit{false}\}}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow\mathit{Bool}\}\rightrightarrows()\,!\,\{\}}$$

Fixing $\mathrm{F}_{\text{eff}} + \mathrm{F}^{\circ}$

We could look at the effect signature of *Choose*:

$$\textbf{handle}\ \overbrace{(\textbf{do}\ \mathit{Choose}\,();\mathit{close}\,f)}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}}\ \textbf{with}\ \overbrace{\{\mathit{Choose}\ \_\ \underset{\mathit{Bool}\to^{\circ}()\,!\,\{\}}{r}\mapsto r\,\mathit{true};r\,\mathit{false}\}}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}\rightrightarrows()\,!\,\{\}}$$

Fixing $F_{eff} + F^{\circ}$

Notice that *close* f uses a linear variable f:

$$\mathbf{handle}\ \overbrace{(\mathbf{do}\ \mathit{Choose}\ (); \mathit{close}\ \underbrace{f}_{f:\mathit{End}\in\Gamma})}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}}\ \mathbf{with}\ \overbrace{\{\mathit{Choose}\ \_\ \underbrace{r}_{\mathit{Bool}\to^{\circ}()\,!\,\{\}} \mapsto r\ \mathit{true}; r\ \mathit{false}\}}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}\rightrightarrows()\,!\,\{\}}$$

Fixing $\mathsf{F}_{\text{eff}} + \mathrm{F}^{\circ}$

Notice that *close* f uses a linear variable f:

$$\textbf{handle}\ (\overbrace{\textbf{do}\ \mathit{Choose}\ ();\ \mathit{close}\ \underbrace{f}_{f:\mathit{End}\in\Gamma}}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}})\ \textbf{with}\ \{\overbrace{\mathit{Choose}\ \_\ \underbrace{r}_{\mathit{Bool}\to^{\circ}()\,!\,\{\}} \mapsto r\ \mathit{true}; r\ \mathit{false}}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}\rightrightarrows()\,!\,\{\}}\}$$

Core idea: add linearity annotations on effect signatures, and track the linearity information while typing.

Fixing $\mathrm{F}_{\mathrm{eff}} + \mathrm{F}^{\circ}$

Notice that *close* f uses a linear variable f:

$$\mathbf{handle}\ \overbrace{(\mathbf{do}\ \mathit{Choose}\ ();\ \mathit{close}\ \underbrace{f}_{f:\mathit{End}\in\Gamma})}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}}\ \mathbf{with}\ \overbrace{\{\mathit{Choose}\ \_\ \underbrace{r}_{\mathit{Bool}\to^{\circ}()\,!\,\{\}} \mapsto r\ \mathit{true}; r\ \mathit{false}\}}^{()\,!\,\{\mathit{Choose}:()\twoheadrightarrow^{\circ}\mathit{Bool}\}\rightrightarrows()\,!\,\{\}}$$

Core idea: add linearity annotations on effect signatures, and track the linearity information while typing.

The linearity Y in $\mathit{Choose} : () \twoheadrightarrow^{Y} \mathit{Bool}$ reflects *control-flow linearity*, i.e. the usage restriction on its context / continuation.

Duality between value linearity and control-flow linearity

For $V : (A : \mathrm{Type}^Y)$, Y restricts the linearity of *the value itself*

- $Y = \circ$
  - V is guaranteed to be used linearly
  - V may contain linear resources
- $Y = \bullet$
  - no guarantee on the usage of V
  - V must not contain linear resources

Duality between value linearity and control-flow linearity

For $V : (A : \mathsf{Type}^Y)$, Y restricts the linearity of *the value itself*

- $Y = \circ$
  - V is guaranteed to be used linearly
  - V may contain linear resources
- $Y = \bullet$
  - no guarantee on the usage of V
  - V must not contain linear resources

less restriction on itself ($\mathsf{Type}^\bullet \leq \mathsf{Type}^\circ$)

Duality between value linearity and control-flow linearity

For $\mathbf{do}\ \ell\, V : A\,!\,\{\ell : A' \twoheadrightarrow^{Y} B'\}$, Y restricts the linearity of *its context*

- $Y = \circ$
  - ℓ is guaranteed to be handled linearly
  - ℓ's continuation may contain linear resources
- $Y = \bullet$
  - no guarantee on the handling of ℓ
  - ℓ's continuation must not contain linear resources

Duality between value linearity and control-flow linearity

For $\textbf{do}\ \ell\, V : A\,!\,\{\ell : A' \twoheadrightarrow^{Y} B'\}$, Y restricts the linearity of *its context*

- $Y = \circ$
  - ℓ is guaranteed to be handled linearly
  - ℓ's continuation may contain linear resources
- $Y = \bullet$
  - no guarantee on the handling of ℓ
  - ℓ's continuation must not contain linear resources

more restriction on its context

Duality between value linearity and control-flow linearity

For $\mathbf{do}\ \ell\, V : A\,!\,\{\ell : A' \twoheadrightarrow^{Y} B'\}$, Y restricts the linearity of *its context*

- $Y = \circ$
  - ℓ is guaranteed to be handled linearly
  - ℓ's continuation may contain linear resources
- $Y = \bullet$
  - no guarantee on the handling of ℓ
  - ℓ's continuation must not contain linear resources

more restriction on its context

However, we cannot upcast $\ell : A' \twoheadrightarrow^{\circ} B'$ to $\ell : A' \twoheadrightarrow^{\bullet} B'$ because it would break the safety of handling.

Duality between value linearity and control-flow linearity

Instead, we can upcast the kind of row types.

For $M : A\,!\,\{(R : \mathrm{Row}_{\emptyset}{}^{Y})\}$, Y restricts the linearity of *its context*

- $Y = \circ$
  - operations in M are guaranteed to be handled linearly
  - M's continuation may contain linear resources
- $Y = \bullet$
  - no guarantee on the handling of operations in M
  - M's continuation must not contain linear resources

more restriction on its context

Duality between value linearity and control-flow linearity

Instead, we can upcast the kind of row types.

For $M : A \,!\, \{(R : \mathrm{Row}_{\emptyset}{}^{Y})\}$, Y restricts the linearity of *its context*

- $Y = \circ$
  - operations in M are guaranteed to be handled linearly
  - M's continuation may contain linear resources
- $Y = \bullet$
  - no guarantee on the handling of operations in M
  - M's continuation must not contain linear resources

more restriction on its context

$$\frac{}{\vdash \bullet \leq \circ} \qquad \frac{\vdash Y \leq Y'}{\vdash \mathrm{Type}^{Y} \leq \mathrm{Type}^{Y'}} \qquad \frac{\vdash Y' \leq Y}{\vdash \mathrm{Presence}^{Y} \leq \mathrm{Presence}^{Y'}} \qquad \frac{\vdash Y' \leq Y}{\vdash \mathrm{Row}_{\mathcal{L}}{}^{Y} \leq \mathrm{Row}_{\mathcal{L}}{}^{Y'}}$$

Tracking Control-Flow Linearity

The evaluation context tells us that continuations consist of only *sequencing* and *handling*.

$$\text{E-OP} \quad \textbf{handle}\ \mathcal{E}[\textbf{do}\ \ell\ V]\ \textbf{with}\ H \leadsto N[V/p, (\lambda y.\textbf{handle}\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H)/r]$$
$$\text{where } \ell \notin \mathsf{bl}(\mathcal{E}) \text{ and } (\ell\ p\ r \mapsto N) \in H$$

$$\text{Evaluation context } \mathcal{E} ::= [\ ] \mid \textbf{let}\ x \leftarrow \mathcal{E}\ \textbf{in}\ N \mid \textbf{handle}\ \mathcal{E}\ \textbf{with}\ H$$

Tracking Control-Flow Linearity

The evaluation context tells us that continuations consist of only *sequencing* and *handling*.

$$\textsc{E-Op}\quad \textbf{handle}\ \mathcal{E}[\textbf{do}\ \ell\ V]\ \textbf{with}\ H \rightsquigarrow N[V/p, (\lambda y.\textbf{handle}\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H)/r]$$
$$\text{where } \ell \notin \mathsf{bl}(\mathcal{E}) \text{ and } (\ell\ p\ r \mapsto N) \in H$$

$$\text{Evaluation context } \mathcal{E} ::= [\ ] \mid \textbf{let}\ x \leftarrow \mathcal{E}\ \textbf{in}\ N \mid \textbf{handle}\ \mathcal{E}\ \textbf{with}\ H$$

As deep handlers are always recursive, they cannot use any linear resource.

T-Handler

$$C = A\,!\,\{(\ell_i : A_i \twoheadrightarrow^{Y_i} B_i)_i; R\} \qquad D = B\,!\,\{(\ell_i : P)_i; R\}$$
$$H = \{\textbf{return}\ x \mapsto M\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i$$
$$\frac{\underset{\text{all types in }\Gamma\text{ are unlimited}}{\Delta \vdash \Gamma : \bullet} \qquad \Delta; \Gamma, x : A \vdash M : D \qquad [\Delta; \Gamma, p_i : A_i, r_i : B_i \to^{Y_i} D \vdash N_i : D]_i}{\Delta; \Gamma \vdash H : C \rightrightarrows D}$$

Tracking Control-Flow Linearity (Cont.)

Sequencing has a real influence on control-flow linearity.

We can make use of the kinding relation of row types:

T-SeqEq

$$\frac{\Delta;\Gamma_1 \vdash M : A \,!\, \{R\} \qquad \Delta;\Gamma_2, x : A \vdash N : B \,!\, \{R\} \qquad \Delta \vdash (\Gamma_2, x : A) : Y \qquad \Delta \vdash R : \mathrm{Row}^{Y}}{\Delta;\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : B \,!\, \{R\}}$$

$Y{=}\circ$: $(\Gamma_2, x : A)$ may contain linear vars

$Y{=}\bullet$: $(\Gamma_2, x : A)$ only contains unlimited vars

$Y{=}\circ$: R only contains linear ops ($\twoheadrightarrow^{\circ}$)

$Y{=}\bullet$: R may contain unlimited ops ($\twoheadrightarrow^{\bullet}$)

Sound, but not Precise Enough?

$$\frac{\cdot;\cdot \vdash \mathbf{do}\ \mathit{Choose}\,() : ()\,!\,\{R\} \qquad \cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Close}\,f : ()\,!\,\{R\} \qquad \cdot \vdash (f : \mathit{End}) : \circ \qquad \cdot \vdash R : \mathrm{Row}_{\emptyset}{}^{\circ}}{\cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Choose}\,(); \mathbf{do}\ \mathit{Close}\,f : ()\,!\,\{R\}}$$

Sound, but not Precise Enough?

$$\frac{\cdot;\cdot \vdash \mathbf{do}\ \mathit{Choose}\ () : ()\ !\ \{R\} \qquad \cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Close}\ f : ()\ !\ \{R\} \qquad \cdot \vdash (f : \mathit{End}) : \circ \qquad \cdot \vdash R : \mathrm{Row}_{\emptyset}{}^{\circ}}{\cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Choose}\ (); \mathbf{do}\ \mathit{Close}\ f : ()\ !\ \{R\}}$$

$R = \{\mathit{Choose} : () \twoheadrightarrow^{\circ} \mathit{Bool}; \mathit{Close} : \mathit{End} \twoheadrightarrow^{\circ} ()\}$ is well-typed but too restrictive

Sound, but not Precise Enough?

$$\frac{\cdot;\cdot \vdash \mathbf{do}\ \mathit{Choose}\,() : ()\,!\,\{R\} \qquad \cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Close}\, f : ()\,!\,\{R\} \qquad \cdot \vdash (f : \mathit{End}) : \circ \qquad \cdot \vdash R : \mathrm{Row}_{\emptyset}{}^{\circ}}{\cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Choose}\,();\mathbf{do}\ \mathit{Close}\, f : ()\,!\,\{R\}}$$

$R = \{\mathit{Choose} : () \twoheadrightarrow^{\circ} \mathit{Bool}; \mathit{Close} : \mathit{End} \twoheadrightarrow^{\circ} ()\}$ is well-typed but too restrictive

$R = \{\mathit{Choose} : () \twoheadrightarrow^{\circ} \mathit{Bool}; \mathit{Close} : \mathit{End} \twoheadrightarrow^{\bullet} ()\}$ is more precise but ill-typed

Sound, but not Precise Enough?

$$\frac{\cdot;\cdot \vdash \mathbf{do}\ \mathit{Choose}\ () : ()\ !\ \{R\} \qquad \cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Close}\ f : ()\ !\ \{R\} \qquad \cdot \vdash (f : \mathit{End}) : \circ \qquad \cdot \vdash R : \mathrm{Row}_{\emptyset}{}^{\circ}}{\cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Choose}\ (); \mathbf{do}\ \mathit{Close}\ f : ()\ !\ \{R\}}$$

$R = \{\mathit{Choose} : () \twoheadrightarrow^{\circ} \mathit{Bool}; \mathit{Close} : \mathit{End} \twoheadrightarrow^{\circ} ()\}$ is well-typed but too restrictive

$R = \{\mathit{Choose} : () \twoheadrightarrow^{\circ} \mathit{Bool}; \mathit{Close} : \mathit{End} \twoheadrightarrow^{\bullet} ()\}$ is more precise but ill-typed

$$\frac{\cdot;\cdot \vdash \mathbf{do}\ \mathit{Choose}\ () : ()\ !\ \{R_1\} \qquad \cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Close}\ f : ()\ !\ \{R_2\} \qquad \cdot \vdash (f : \mathit{End}) : \circ \qquad \cdot \vdash R_1 : \mathrm{Row}_{\emptyset}{}^{\circ} \qquad R_1 \leqslant R_2}{\cdot; f : \mathit{End} \vdash \mathbf{do}\ \mathit{Choose}\ (); \mathbf{do}\ \mathit{Close}\ f : ()\ !\ \{R_2\}}$$

$R_1 = \{\mathit{Choose} : () \twoheadrightarrow^{\circ} \mathit{Bool}\}, R_2 = \{\mathit{Choose} : () \twoheadrightarrow^{\circ} \mathit{Bool}; \mathit{Close} : \mathit{End} \twoheadrightarrow^{\bullet} ()\}.$

More Precise Typing Rule for Sequencing

T-SeqSub

$$\frac{\Delta;\Gamma_1 \vdash M : A\,!\,\{R_1\} \qquad \Delta;\Gamma_2, x : A \vdash N : B\,!\,\{R_2\} \qquad \Delta \vdash (\Gamma_2, x : A) : Y \qquad \Delta \vdash R : \mathrm{Row}^{Y} \qquad R_1 \leqslant R_2}{\Delta;\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : B\,!\,\{R_2\}}$$

More Precise Typing Rule for Sequencing

$$\textsc{T-SeqSub}\quad \frac{\Delta; \Gamma_1 \vdash M : A\,!\,\{R_1\} \qquad \Delta; \Gamma_2, x : A \vdash N : B\,!\,\{R_2\} \qquad \Delta \vdash (\Gamma_2, x : A) : Y \qquad \Delta \vdash R : \mathrm{Row}^Y \qquad R_1 \leqslant R_2}{\Delta; \Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : B\,!\,\{R_2\}}$$

Row subtyping relation $\boxed{R_1 \leqslant R_2}$

$$\frac{}{R \leqslant R} \qquad \frac{R_1 \leqslant R_2 \qquad R_2 \leqslant R_3}{R_1 \leqslant R_3} \qquad \frac{}{\cdot \leqslant \mu} \qquad \frac{R_1 \leqslant R_2}{\ell : \mathrm{Abs}; R_1 \leqslant \ell : P; R_2} \qquad \frac{R_1 \leqslant R_2}{\ell : P; R_1 \leqslant \ell : P; R_2}$$

More Precise Typing Rule for Sequencing

$$\textsc{T-SeqSub}\quad \frac{\Delta;\Gamma_1 \vdash M : A\,!\,\{R_1\} \qquad \Delta;\Gamma_2, x : A \vdash N : B\,!\,\{R_2\} \qquad \Delta \vdash (\Gamma_2, x : A) : Y \qquad \Delta \vdash R : \mathsf{Row}^Y \qquad R_1 \leqslant R_2}{\Delta;\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : B\,!\,\{R_2\}}$$

Row subtyping relation $\boxed{R_1 \leqslant R_2}$

$$\frac{}{R \leqslant R} \qquad \frac{R_1 \leqslant R_2 \qquad R_2 \leqslant R_3}{R_1 \leqslant R_3} \qquad \frac{}{\cdot \leqslant \mu} \qquad \frac{R_1 \leqslant R_2}{\ell : \mathsf{Abs}; R_1 \leqslant \ell : P; R_2} \qquad \frac{R_1 \leqslant R_2}{\ell : P; R_1 \leqslant \ell : P; R_2}$$

Although it is folklore that row polymorphism can replace row subtyping to some extent (especially for effect types), in settings like tracking control-flow linearity, a combination of them is better.

F°_{eff} Metatheory

Standard progress and preservation.

$\mathrm{F}^{\circ}_{\mathrm{eff}}$ Metatheory

Standard progress and preservation.

Lemma (Unlimited values are unlimited)

If $\Delta; \Gamma \vdash V : A$ *and* $\Delta \vdash A : \bullet$*, then* $\Delta \vdash \Gamma : \bullet$.

$\mathrm{F}^{\circ}_{\mathrm{eff}}$ Metatheory

Standard progress and preservation.

Lemma (Unlimited values are unlimited)

If $\Delta; \Gamma \vdash V : A$ *and* $\Delta \vdash A : \bullet$, *then* $\Delta \vdash \Gamma : \bullet$.

Lemma (Unlimited operations are unlimited)

If $\Delta; \Gamma \vdash \mathcal{E}[(\mathbf{do}\ \ell\ V)^E] : A \,!\, \{\ell : A' \twoheadrightarrow^{\bullet} B', R\}$ *and* $\ell \notin \mathrm{bl}(\mathcal{E})$, *then there exists* $\Delta \vdash \Gamma = \Gamma_1 + \Gamma_2$ *s.t.* $\Delta \vdash \Gamma_1 : \bullet$ *and* $\Delta; \Gamma_1, y : B_\ell \vdash \mathcal{E}[\mathbf{return}\ y] : A \,!\, \{\ell : A' \twoheadrightarrow^{\bullet} B'; R\}$.

$\mathsf{F}^{\circ}_{\text{eff}}$ Metatheory

Standard progress and preservation.

Lemma (Unlimited values are unlimited)

If $\Delta; \Gamma \vdash V : A$ *and* $\Delta \vdash A : \bullet$, *then* $\Delta \vdash \Gamma : \bullet$.

Lemma (Unlimited operations are unlimited)

If $\Delta; \Gamma \vdash \mathcal{E}[(\mathbf{do}\ \ell\ V)^E] : A\,!\,\{\ell : A' \twoheadrightarrow^{\bullet} B', R\}$ *and* $\ell \notin \mathrm{bl}(\mathcal{E})$, *then there exists* $\Delta \vdash \Gamma = \Gamma_1 + \Gamma_2$ *s.t.* $\Delta \vdash \Gamma_1 : \bullet$ *and* $\Delta; \Gamma_1, y : B_\ell \vdash \mathcal{E}[\mathbf{return}\ y] : A\,!\,\{\ell : A' \twoheadrightarrow^{\bullet} B'; R\}$.

By further defining a linearity-aware semantics, we can show that every linear value is used exactly once during evaluation.

Theorem (Evaluation linearity)

If M *is proper and* $M \overset{\mathcal{C}}{\underset{\mathcal{D}}{\leadsto}} N$, *then* N *is also proper and* $\mathscr{L}(M) \uplus \mathscr{L}(\mathcal{C}) = \mathscr{L}(N) \uplus \mathscr{L}(\mathcal{D})$.

What about Type Inference ?

Challenges of $\mathsf{F}^{\circ}_{\mathrm{eff}}$ type inference:

What about Type Inference ?

Challenges of $\mathsf{F}^{\circ}_{\text{eff}}$ type inference:

- First-class polymorphism

What about Type Inference ?

Challenges of $\mathsf{F}^{\circ}_{\mathrm{eff}}$ type inference:

- First-class polymorphism
- Linear types and subkinding

What about Type Inference ?

Challenges of $\mathsf{F}^{\circ}_{\mathrm{eff}}$ type inference:

- First-class polymorphism
- Linear types and subkinding
- Row subtyping

What about Type Inference ?

Challenges of F°_{eff} type inference:

- First-class polymorphism $\Rightarrow$ prenex polymorphism
- Linear types and subkinding
- Row subtyping

What about Type Inference ?

Challenges of F°_{eff} type inference:

- First-class polymorphism ⇒ prenex polymorphism
- Linear types and subkinding ⇒ qualified types (Quill[5])
- Row subtyping

[5] Morris, “The Best of Both Worlds: Linear Functional Programming without Compromise”, 2016.

What about Type Inference ?

Challenges of F°_{eff} type inference:

- First-class polymorphism $\Rightarrow$ prenex polymorphism
- Linear types and subkinding $\Rightarrow$ qualified types (QUILL[5])
- Row subtyping $\Rightarrow$ qualified types (ROSE[6])

[5] Morris, "The Best of Both Worlds: Linear Functional Programming without Compromise", 2016.
[6] Morris and McKinna, "Abstracting Extensible Data Types: Or, Rows by Any Other Name", 2019.

What about Type Inference ?

Challenges of F°_{eff} type inference:

- First-class polymorphism $\Rightarrow$ prenex polymorphism
- Linear types and subkinding $\Rightarrow$ qualified types (Quill[5])
- Row subtyping $\Rightarrow$ qualified types (Rose[6])

Q°_{eff} : A ML-style variant of F°_{eff} based on qualified types with

- full type inference without any type annotations
- accurate tracking of control-flow linearity (even more accurate than F°_{eff})

[5] Morris, "The Best of Both Worlds: Linear Functional Programming without Compromise", 2016.
[6] Morris and McKinna, "Abstracting Extensible Data Types: Or, Rows by Any Other Name", 2019.

Future Work

- Shallow handlers:
  - linear shallow handlers can also introduce linear resources into continuations with a more complex behaviour than sequencing;
  - not entirely sure how to track it most accurately.

Future Work

- Shallow handlers:
  - linear shallow handlers can also introduce linear resources into continuations with a more complex behaviour than sequencing;
  - not entirely sure how to track it most accurately.
- FreezeML(X):
  - Links supports first-class polymorphism using FreezeML[7];
  - non-trivial to extend FreezeML with qualified types.

[7] Emrich et al., “FreezeML: Complete and Easy Type Inference for First-Class Polymorphism”, 2020.

Future Work

- Shallow handlers:
  - linear shallow handlers can also introduce linear resources into continuations with a more complex behaviour than sequencing;
  - not entirely sure how to track it most accurately.
- FreezeML(X):
  - Links supports first-class polymorphism using FreezeML[7];
  - non-trivial to extend FreezeML with qualified types.
- Other classifications of effects:
  - besides linear ($\twoheadrightarrow^{\circ}$) and unlimited effects ($\twoheadrightarrow^{\bullet}$), our method can also be used for other classifications, like algebraic effects vs. higher-order effects.

[7] Emrich et al., "FreezeML: Complete and Easy Type Inference for First-Class Polymorphism", 2020.

Thank you!

$F_{eff} + F^{\circ} = F^{\circ}_{eff}$

		Kinds K ::=
Value types	$A, B ::= \alpha \mid A \to^{Y} C \mid \forall^{Y} \alpha^{K}.C$	Type^{Y}
Computation types	$C, D ::= A\,!\,E$	Comp
Effect types	$E ::= \{R\}$	Effect
Row types	$R ::= \ell : P; R \mid \mu \mid \cdot$	$\mathsf{Row}^{Y}_{\mathcal{L}}$
Presence types	$P ::= \mathsf{Abs} \mid A \twoheadrightarrow^{Y} B \mid \theta$	$\mathsf{Presence}^{Y}$
Handler types	$F ::= C \rightrightarrows D$	Handler
Types	$T ::= A \mid R \mid P$	
Label sets	$\mathcal{L} ::= \emptyset \mid \{\ell\} \uplus \mathcal{L}$	
Linearity	$Y ::= \bullet \mid \circ$	
Type contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$	
Kind contexts	$\Delta ::= \cdot \mid \Delta, \alpha : K$	
Values	$V, W ::= x \mid \lambda^{Y} x^{A}.M \mid \Lambda^{Y} \alpha^{K}.M$	
Computations	$M, N ::= V\,W \mid V\,T \mid (\mathbf{return}\ V)^{E} \mid (\mathbf{do}\ \ell\ V)^{E}$ $\mid \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \mid \mathbf{handle}\ M\ \mathbf{with}\ H$	
Handlers	$H ::= \{\mathbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H$	

15

Duality between value linearity and control-flow linearity

Instead, we can upcast the kind of row type.

For $M : A\,!\,\{(R : \mathsf{Row}_{\emptyset}{}^{Y})\}$, Y restricts the linearity of *its context*

- $Y = \circ$
  - operations in M are guaranteed to be handled linearly
  - M's continuation may contain linear resources
- $Y = \bullet$
  - no guarantee on the handling of operations in M
  - M's continuation must not contain linear resources

more restriction on its context

$$\frac{}{\vdash \bullet \leq \circ} \qquad \frac{\vdash Y \leq Y'}{\vdash \mathsf{Type}^{Y} \leq \mathsf{Type}^{Y'}} \qquad \frac{\vdash Y' \leq Y}{\vdash \mathsf{Presence}^{Y} \leq \mathsf{Presence}^{Y'}} \qquad \frac{\vdash Y' \leq Y}{\vdash \mathsf{Row}_{\mathcal{L}}{}^{Y} \leq \mathsf{Row}_{\mathcal{L}}{}^{Y'}}$$

14

More Precise Typing Rule for Sequencing

T-SeqSub

$$\frac{\Delta;\Gamma_1 \vdash M : A\,!\,\{R_1\} \qquad \Delta;\Gamma_2, x : A \vdash N : B\,!\,\{R_2\} \qquad \Delta \vdash (\Gamma_2, x : A) : Y \qquad \Delta \vdash R : \mathsf{Row}^{Y} \qquad R_1 \leq R_2}{\Delta;\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : B\,!\,\{R_2\}}$$

Row subtyping relation $\boxed{R_1 \leq R_2}$

$$\frac{}{R \leq R} \qquad \frac{R_1 \leq R_2 \qquad R_2 \leq R_3}{R_1 \leq R_3} \qquad \frac{}{\cdot \leq \mu} \qquad \frac{R_1 \leq R_2}{\ell : \mathsf{Abs}; R_1 \leq \ell : P; R_2} \qquad \frac{R_1 \leq R_2}{\ell : P; R_1 \leq \ell : P; R_2}$$

Although it is folklore that row polymorphism can replace row subtyping to some extent (especially for effect types), in settings like tracking control-flow linearity, a combination of them is better.

19

What about Type Inference ?

Challenges of F°_{eff} type inference:

- First-class polymorphism $\Rightarrow$ prenex polymorphism
- Linear types and subkinding $\Rightarrow$ qualified types (Quill[5])
- Row subtyping $\Rightarrow$ qualified types (Rose[6])

Q°_{eff} : A ML-style variant of F°_{eff} based on qualified types with

- full type inference without any type annotations
- accurate tracking of control-flow linearity (even more accurate than F°_{eff})

[5] Morris, "The Best of Both Worlds: Linear Functional Programming without Compromise", 2016.
[6] Morris and McKinna, "Abstracting Extensible Data Types: Or, Rows by Any Other Name", 2019.

21

F°_{eff} Kinding Rules for Value Types

Kinding relation $\boxed{\Delta \vdash A : K}$

$$\frac{}{\vdash \bullet \leq \circ} \qquad \frac{\vdash Y \leq Y'}{\vdash \mathsf{Type}^{Y} \leq \mathsf{Type}^{Y'}}$$

$$\textsc{K-TyVar}\ \frac{}{\Delta, \alpha : K \vdash \alpha : K} \qquad \textsc{K-Forall}\ \frac{\Delta, \alpha : K \vdash C : \mathsf{Comp}}{\Delta \vdash \forall^{Y} \alpha^{K}. C : \mathsf{Type}^{Y}}$$

$$\textsc{K-Fun}\ \frac{\Delta \vdash A : \mathsf{Type}^{Y'} \quad \Delta \vdash C : \mathsf{Comp}}{\Delta \vdash A \to^{Y} B : \mathsf{Type}^{Y}} \qquad \textsc{K-Upcast}\ \frac{\Delta \vdash T : K \quad \vdash K \leq K'}{\Delta \vdash T : K'}$$

Extend to contexts $\boxed{\Delta \vdash \Gamma : Y}$

- $Y = \circ : \Gamma$ *may* contain linear variables (because of K-Upcast)
- $Y = \bullet : \Gamma$ only contains unlimited variables

Context splitting $\boxed{\Delta \vdash \Gamma = \Gamma_1 + \Gamma_2}$

- Variables with unlimited types appear in both Γ_1 and Γ_2
- Variables with linear types only appear in one of them

F°_{eff} Kinding Rules for Other Types

K-Effect
$$\frac{\Delta \vdash R : \mathsf{Row}^{Y}_{\emptyset}}{\Delta \vdash \{R\} : \mathsf{Effect}}$$

K-Comp
$$\frac{\Delta \vdash A : \mathsf{Type}^{Y} \qquad \Delta \vdash E : \mathsf{Effect}}{\Delta \vdash A \,!\, E : \mathsf{Comp}}$$

K-Handler
$$\frac{\Delta \vdash C : \mathsf{Comp} \qquad \Delta \vdash D : \mathsf{Comp}}{\Delta \vdash C \rightrightarrows D : \mathsf{Handler}}$$

K-Absent
$$\frac{}{\Delta \vdash \mathsf{Abs} : \mathsf{Presence}^{Y}}$$

K-Present
$$\frac{}{\Delta \vdash A \twoheadrightarrow^{Y} B : \mathsf{Presence}^{Y}}$$

K-EmptyRow
$$\frac{}{\Delta \vdash \cdot : \mathsf{Row}_{\mathcal{L}}{}^{Y}}$$

K-ExtendRow
$$\frac{\Delta \vdash P : \mathsf{Presence}^{Y} \qquad \Delta \vdash R : \mathsf{Row}_{\mathcal{L} \uplus \{\ell\}}{}^{Y}}{\Delta \vdash \ell : P; R : \mathsf{Row}_{\mathcal{L}}{}^{Y}}$$

F°_{eff} Metatheory (Cont.)

Definition (Properness)

A well-typed computation M or value V is proper if and only if,

1. for every sub-values W in it, if W *has* some type A which can be given kind $\mathsf{Type}^{\bullet}$, then $\mathscr{L}(W) = \emptyset$;
2. for every sub-computation N of form $\mathcal{E}[\mathbf{do}\ \ell\ V]$ where $\ell \notin \mathsf{bl}(\mathcal{E})$ in it, if N *has* some effect type $\{\ell : A_\ell \twoheadrightarrow^{\bullet} B_\ell; \ldots\}$, then $\mathscr{L}(\mathcal{E}) = \emptyset$.

Q°_{eff} Qualified Types

Row types	$R ::= \mu \mid \overline{\ell : A \twoheadrightarrow^{Y} B}$
Linearity	$Y ::= \phi \mid \bullet \mid \circ$
Types	$\tau ::= A \mid R \mid Y$
Predicates	$\pi ::= \underbrace{\tau_1 \leq \tau_2}_{\text{only compare linearity}} \mid \underbrace{R_1 \otimes R_2}_{\text{only compare label sets}} \mid R_1 \odot R_2 \sim R$
Qualified types	$\rho ::= A \mid \pi \Rightarrow \rho$
Type schemes	$\sigma ::= \rho \mid \forall \alpha.\sigma$

Back to the "print then close" example:

$$\begin{aligned}&\textbf{do}\ \textit{Print}\,\text{"42"}; \textbf{do}\ \textit{Close}\ f\ :\\ &\quad \forall \mu\, \phi_1\, \phi_2.((\textit{Print} : \phi_1) \otimes \mu, (\textit{Close} : \phi_2) \otimes \mu, \textit{File} \leq \phi_1) \Rightarrow ()\,!\,\{\mu\}\end{aligned}$$

As we know *File* is a linear type, we can further simplify it to:

$$\textbf{do}\ \textit{Print}\,\text{"42"}; \textbf{do}\ \textit{Close}\ f\ : \forall \mu\, \phi.((\textit{Print} : \circ) \otimes \mu, (\textit{Close} : \phi) \otimes \mu) \Rightarrow ()\,!\,\{\mu\}$$

Q°_{eff} Typing Rules

Typing relation $\boxed{P \mid \Gamma \vdash V : A}$ $\boxed{P \mid \Gamma \vdash M : C}$ $\boxed{P \mid \Gamma \vdash H : C \rightrightarrows D}$

Q-Abs

$$\frac{P \mid \Gamma, x : A \vdash M : C \qquad P \vdash \Gamma \leq Y}{P \mid \Gamma \vdash \lambda x.M : A \to^{Y} C}$$

"any type in Γ" $\leq Y$
$Y = \bullet$: all vars in Γ are unlimited
$Y = \circ$: essentially no restriction
$Y = \phi$: collect the constraint in P

Q-Handler

$$\frac{\begin{array}{c} H = \{\textbf{return}\ x \mapsto M\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\ D = B\,!\,\{R_2\} \qquad P \mid \Gamma, x : A \vdash M : D \\ [P \mid \Gamma, p_i : A_i, r_i : B_i \to^{Y_i} D \vdash N_i : D]_i \\ P \vdash \Gamma \leq \bullet \\ P \Rightarrow (\ell_i : A_i \twoheadrightarrow^{Y_i} B_i)_i \odot R \sim R_1 \qquad P \Rightarrow R \oslash R_2 \end{array}}{P \mid \Gamma \vdash H : A\,!\,\{R_1\} \rightrightarrows B\,!\,\{R_2\}}$$

$P \vdash \Gamma \leq \bullet$: all vars in Γ are unlimited
$P \Rightarrow (\ell_i : A_i \twoheadrightarrow^{Y_i} B_i)_i \odot R \sim R_1$: combination of $(\ell_i)_i$ and R
$P \Rightarrow R \oslash R_2$: R_2 contains R

Q-Seq

$$\frac{\begin{array}{c} P \mid \Gamma_1, \Gamma \vdash M : A\,!\,\{R_1\} \qquad P \mid \Gamma_2, \Gamma, x : A \vdash N : B\,!\,\{R_2\} \\ P \vdash \Gamma \leq \bullet \qquad P \Rightarrow R_1 \oslash R_2 \qquad P \vdash (\Gamma_2, x : A) \leq R_1 \end{array}}{P \mid \Gamma_1, \Gamma_2, \Gamma \vdash \textbf{let}\ x \leftarrow M\ \textbf{in}\ N : B\,!\,\{R_2\}}$$

$P \vdash \Gamma \leq \bullet$: all vars in Γ are unlimited
$P \Rightarrow R_1 \oslash R_2$: R_2 contains R_1
$P \vdash (\Gamma_2, x : A) \leq R_1$: "any type in $(\Gamma_2, x : A)$" $\leq$ "any label in R_1"
$R_1 = (\ell_i : Y_i)_i$: $[P \vdash (\Gamma_2, x : A) \leq Y_i]_i$
$R_1 = \mu$: collect the constraint in P

Q°_{eff} Type Inference

Almost standard Hindley-Milner type inference with qualified types.

Metatheory: Standard soundness and completeness.

Theorem (Soundness)

If $\theta;\Gamma \vdash V : A \dashv \theta', P, \Sigma$, *then* $\theta' P \mid \theta'(\Gamma|_{\Sigma}) \vdash V : \theta' A$. *The same applies to computation and handler typing.*

Theorem (Completeness)

If $P \mid \theta\Gamma \vdash V : A$, *then* $\iota;\Gamma \vdash V : A' \dashv \theta', Q, \Sigma$ *and there exists* θ'' *such that* $A = \theta''\theta' A'$, $P \Rightarrow \theta''\theta' Q$, *and* $\theta = (\theta''\theta')|_{\Gamma}$. *The same applies to computation and handler typing.*

Constraint solving? A seemingly correct graph algorithm for checking and simplifying constraints.

$\mathsf{Q}^{\circ}_{\mathsf{eff}}$ More Example

Consider the following function:

$$\lambda^{\bullet} f.\lambda^{\bullet} g.f\,();g\,()$$

The type inference of $\mathsf{F}^{\circ}_{\mathsf{eff}}$ infers the following principal type:

$$\begin{aligned}&\forall \alpha_1\,\alpha_2\,\mu_1\,\mu_2\,\phi_1\,\phi_2.(\phi_2 \leq \mu_1, \mu_1 \leqslant \mu_2)\\ &\quad\Rightarrow (() \to^{\phi_1} \alpha_1\,!\,\{\mu_1\}) \to^{\bullet} (() \to^{\phi_2} \alpha_2\,!\,\{\mu_2\}) \to^{\bullet} \alpha_2\,!\,\{\mu_2\}\end{aligned}$$

While in $\mathsf{F}^{\circ}_{\mathsf{eff}}$, the subtyping relation $\mu_1 \leqslant \mu_2$ requires $\mu_1 = \mu_2$, which is more restrictive.